Most games have the element of chance in them somewhere. Children's games tend to be largely chance oriented, depending upon throwing dice or spinning a pointer. Adult games eliminate some of the chance elements and tend to pit the skills of the players against each other. Gambling games are largely chance dependent although skillful players know enough of the laws of probability and human psychology to beat other, less informed opponents fairly consistently. One of the more amusing uses of computers is in programming them to play games. This presumes the ability of the computer to introduce the element of chance into the game.

Statistical analysis often requires that a program be able to, upon demand, produce a number that can be used to test the analysis. This number should be totally random in nature, so that each random number produced is different from the others. Even this is an oversimplification since nature occasionally produces duplicates; a true random number will occasionally be repeated. By the same token a number like 5,555,555,555 is just as random as any other; it is just easier to remember.

The longest run of even numbers on the Monte Carlo roulette tables is 28 consecutive passes; the chances of this occurring are 1 in 89,478,485. The fact that it has happened once does not mean that it will not happen again tomorrow. This makes it very difficult to asses the randomness of events.

Generally speaking there are two easy ways to produce random numbers from your computer. Assume that you are interested in producing a random number 8-bits in length. One way to accomplish this is to take whatever is in the accumulator, add 1, check the keyboard, and so on. This incrementing process happens so fast that the accumulator runs through all of the 8-bit numbers possible thousands of times a second. When a key closure is found, the contents of the accumulator at that instant become the random number. This method works only because the

rate at which numbers are produced in the accumulator is far faster than the speed with which you can press a key. It would be impossible for someone to press the key so fast that the number would necessarily be low (for that matter the user really did not know what was in the accumulator when the process started). The disadvantage to this technique is that when many numbers must be produced, the program gets unnecessarily complicated because of all the keyboard routines, to say nothing of the confusion resulting because the user has to repeatedly press the keys.

A more desireable method is illustrated here. It assumes that the contents of the memory are entirely random when the computer is first turned on. It adds all of the random contents of the RAM memory locations together which in turn produces a random number. Since that number has to be stored somewhere in memory to be useful, the memory is changed each time the program produces a new random number, so that numbers can be produced endlessly without repeating in some pattern. The method is illustrated in Fig. 11-1 which is the flow diagram for the random number generator that we wish to include in the utility subroutines. Enter the program in your computer at location 3070H, studying the flow diagram as you do so.

RANDOM NUMBER GENERATOR

```
0370    E3    XTHL            ;Load return address presently
0371    D5    PUSH   D        ;at the top of stack into the H/L
0372    56    MOV    D, M     ;registers. Save these registers
0373    23    INX    H        ;in the stack in place of the return
0374    33    INX    SP       ;address. Save D/E in stack. Fetch
0375    33    INX    SP       ;the range number that followed the
0376    E3    XTHL            ;CALL to this subroutine to the D
0377    3B    DCX    SP       ;register. Increment H/L to adjust the
0378    3B    DCX    SP       ;return address. Increment stack
0379    E5    PUSH   H        ;and exchange return address and
```

```
037A   21   LXI   H, 0000H     ;H/L. Return stack pointer to first
037B   00                      ;value. Store H/L in stack. Point
037C   00                      ;H/L to first location in RAM.
037D   8E   ADC   M            ;Add contents of memory location
037E   F5   PUSH  PSW          ;to accum. Save accumulator in stack.
037F   7C   MOV   A, H         ;Check to see if last memory loca-
0380   FE   CPI   04H          ;tion. If not, restore accumulator from
0381   04                      ;stack, increment to point to next
0382   CA   JZ    038AH        ;memory location and loop back to
0383   8A                      ;037DH. If it was the last memory
0384   03                      ;location, jump to 038AH.
0385   F1   POP   PSW          ;
0386   23                      ;
0387   C3   JMP   037DH        ;
0388   7D                      ;
0389   03                      ;
038A   F1   POP   PSW          ;Restore accumulator
038B   67   MOV   H, A         ;from stack. Save it in H.
038C   E6   ANI   0FH          ;Zero out four left bits in
038D   0F                      ;accum.
038E   6F   MOV   L, A         ;Save this abbreviated number in L.
038F   7C   MOV   A, H         ;Fetch original number back to accumulator
0390   E6   ANI   F0H          ;and zero out the right-most four bits.
0391   F0                      ;
0392   0F   RRC                ;Rotate accumulator four bit positions so
0393   0F   RRC                ;number is now contained in four
0394   0F   RRC                ;right-most bit positions.
0395   0F   RRC                ;
0396   85   ADD   L            ;Add the contents of accumulator to contents
0397   E6   ANI   0FH          ;of L to form new random number. Zero
0398   0F                      ;out any fifth bit that may have
0399   BA   CMP   D            ;resulted from carry of the ADD L.
```

```
039A    CA    JZ     03A8H       ;If the resulting random number is
039B    A8                       ;within the range specified by the
039C    03                       ;D register, jump to 03A8H to exit
039D    DA    JC     03A8H       ;the subroutine.  If not, the memory
039E    A8                       ;must be changed and the process
039F    03                       ;repeated to get a new random number.
03A0    21    LXI    H, 0000H    ;This is done by rotating the contents
03A1    00                       ;of the accumulator one bit to the
03A2    00                       ;right and storing in 0000H.
03A3    1F    RAR                ;The process of generating a random
03A4    34    MOV    M, A        ;number is then repeated.
03A5    C3    JMP    037AH       ;
03A6    7A                       ;
03A7    03                       ;
03A8    E1    POP    H           ;Exit the subroutine.  Restore
03A9    D1    POP    D           ;the registers and return.
03AA    C9    RET                ;
```

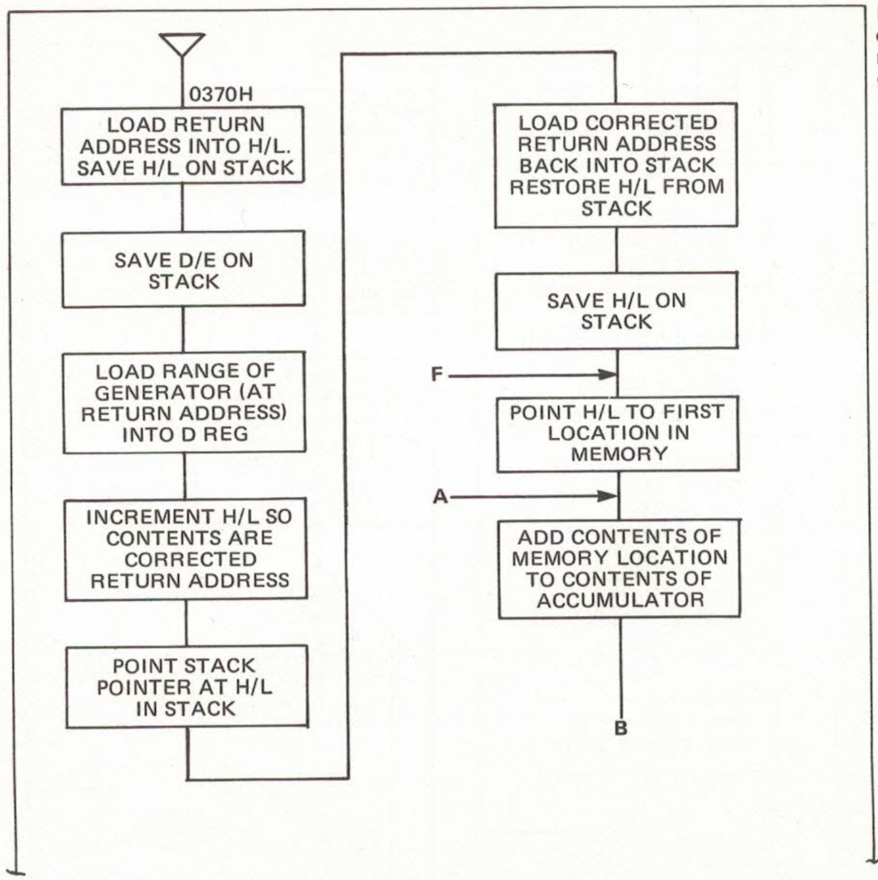
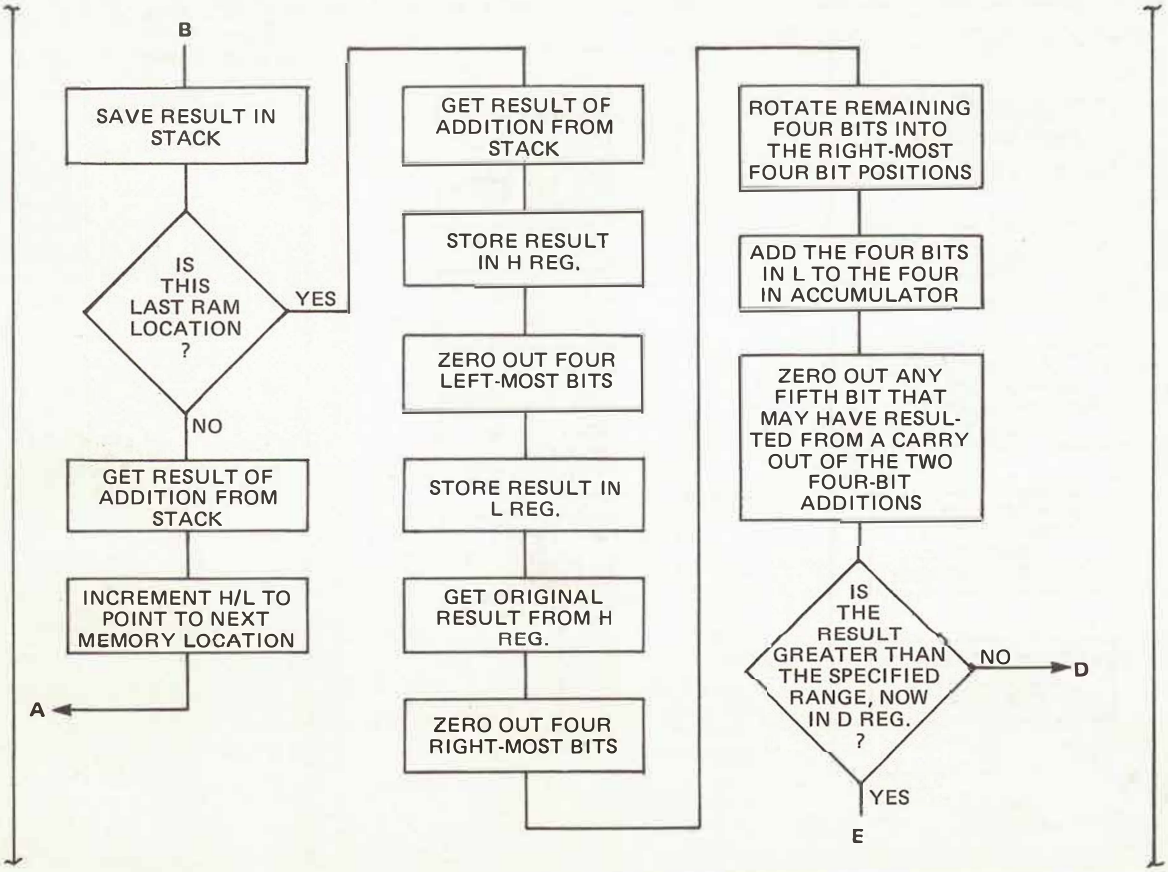
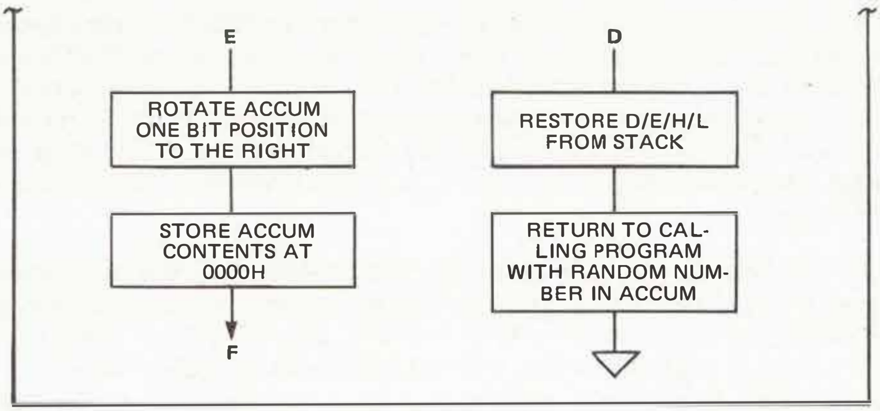


Fig. 11-1 Flow diagram of random number generator.


B
SAVE RESULT IN STACK
IS THIS LAST RAM LOCATION ?
YES
NO
GET RESULT OF ADDITION FROM STACK
INCREMENT H/L TO POINT TO NEXT MEMORY LOCATION
A
GET RESULT OF ADDITION FROM STACK
STORE RESULT IN H REG.
ZERO OUT FOUR LEFT-MOST BITS
STORE RESULT IN L REG.
GET ORIGINAL RESULT FROM H REG.
ZERO OUT FOUR RIGHT-MOST BITS
ROTATE REMAINING FOUR BITS INTO THE RIGHT-MOST FOUR BIT POSITIONS
ADD THE FOUR BITS IN L TO THE FOUR IN ACCUMULATOR
ZERO OUT ANY FIFTH BIT THAT MAY HAVE RESULTED FROM A CARRY OUT OF THE TWO FOUR-BIT ADDITIONS
IS THE RESULT GREATER THAN THE SPECIFIED RANGE, NOW IN D REG. ?
NO
D
YES
E



The program begins by fetching the range of the random number that is specified immediately after the CALL to the subroutine. This is done by executing an XTHL to save the H and L registers in the stack and load the return address into H/L. The return address is actually the address of the location that contains the range of the random numbers. We will need D and E in the subroutine so they are freed with a PUSH D. MOV D, M then causes the range of the random number to be loaded into the D register. The H/L pair is incremented with INX H so that the return address contained there is now correct. We want to reload the corrected address back into the stack but the PUSH D executed earlier has moved the stack pointer two locations beyond the desired spot which now is temporarily storing H and L. We can correct the stack pointer by executing two INX SP instructions. The INX SP, Increment Stack Pointer, operates just as INX H, INX D and INX B. Those instructions treated the register pair as a 16-bit register; the INX SP does the same thing, but the fact that the stack pointer is in reality a 16-bit register makes the instruction easier to understand. An XTHL then swaps the corrected return address in H/L back into the stack in exchange for the original contents of the H and L registers. We must now move

the stack pointer back to the point it was before we adjusted it with two INX SP instructions. This is done by executing a pair of DCX SP instructions which decrement the stack pointer twice. When this has been accomplished, the corrected return address is in the correct spot in the stack; below that is the original contents of the D and E registers placed there with the PUSH D. H and L contain their original contents and D contains the number that represents the specified range of the subroutine. When the subroutine is finally exited, it will be with a number in the accumulator that does not excede this range.

Because we will need the H/L registers in this subroutine, we PUSH them onto the stack now. The H/L registers have been in the stack before this subroutine, but that was a coincidence because of the way we retrieved the range number from the CALLing program. When that number was finally loaded into the D register, the H/L registers had ended up back in their original state, so we must "resave" them now. When this has been done we point H/L to the first location in RAM memory, location 0000H. We are now about to generate the random number. This will be done by adding up all of the contents of all of the RAM memory locations in the computer, a process that takes a small fraction of a second. During that process we will ignore the overflow out of the accumulator so that the result is abbreviated to the eight least significant bits of the true result. That would give us a random number in the range of 0-FF, far in excess of the range we wish. This program limits the range to 0-F. This number will be specified by four bits, so the 8-bit number in the accumulator must be compressed. This could be done by just taking the four right-most bits of the accumulator as the random number, but in an effort to improve the randomness of our number, we will shift the four left-most bits four places to the right and add them to the bits already there. The four-bit result will be our final random number. That addition may result in a carry out into the fifth bit position, so we execute an ANI 0FH to zero out the fifth bit.

We need only to check to see if the random number we have produced is within the specified range, contained in the D register. A CMP D performs the check. This brings us to a common

problem in programming. We wish to determine if one number is larger than another. The CMP or CPI instructions set the zero flag if the two numbers are equal, but give us no indication as to which is larger. For this we have to examine the carry flag. If the number in the accumulator is smaller than the immediate data or the data in the register being compared, the carry flag will be set to 1. In this particular case, we wish to determine if the data in the accumulator, the random number, is equal to or smaller than the range number in the D register. We can check to see if the random number equals the range with a JZ test. A JC, Jump if Carry, tests if the random number is smaller than the range specified by the D register. In either case, the subroutine should be exited and a jump to 03A8H accomplishes that.

If the random number generated is outside of the range specified, we will have to go back and start over. But if we're not careful, we'll just generate the same random number and we'll find ourselves in an endless loop of generating unacceptable random numbers. To prevent this, we modify one byte of memory so that the process of summing all of the memory locations will generate a new number. Since this is a utility subroutine, we have to be very careful which memory location is changed. After all, we wouldn't want to be arbitrarily changing some location which contains useful data! The memory locations that store the data being displayed are useful for this purpose. We need not worry about changing the data here since the display registers are reloaded by the CONVERT subroutine before the display subroutine. So, in order to modify the memory to generate a new random number, we will modify location 0000H, display register 0. This is done by rotating the random number one bit position to the right and storing that number in the memory at location 0000H. This changes the memory so that the next random number will be different. A jump back to 037AH starts the process over. When an acceptable random number is finally generated it will pass the test and exit to 03A8H where a series of POPs and a RET conclude the subroutine.

MASTER-MIND. This is a fascinating game originally designed for two players. One player makes up a "code" by placing a series of pegs in four holes behind a shield so the other player could not see the code word. There were six colors of pegs, and since the code word could contain two or more pegs of the same color, the possible number of code words was extremely large. When the code word had been selected the other player proceded to guess the code. Naturally this would take an extremely long time, if it were not for the fact that as the player makes each guess the other player scores him. These scores serve as clues so that the guesser is able to modify his guesses as he goes and home in on the correct code word. With practice, it is possible to consistently arrive at the correct code word with only four or five guesses.

In the conventional form of the game both code word and guesses are done using pegs. The board upon which play progresses has the code word, hidden behind the shield, at one end with an array of holes across the rest of the board. As each guess is made, it is physically implemented by placing pegs in a row of holes with the colors and positions corresponding to the guess. This provides a record of the guess in case there is a dispute over scoring. As each guess is made, the correct pegs are placed in a new row, so that the number of rows corresponds to the number of guesses made. Scoring is done by the first player, the code maker, checking to see if any of the pegs are in both the right position and are the right color. For each of the pegs in the guess that fit within this category, a small black scoring peg is awarded. If the guess is completely correct, the player will be awarded four black pegs and the game is over. Assuming there are some incorrect pegs, the scorer goes on to consider pegs that are the right color but in the wrong position. For each peg in this category, the guesser is awarded one white scoring peg.

Scoring is somewhat confusing at first. To avoid mistakes be sure the following rules are kept in mind. Score with the black scoring pegs first. This is easy since a colored peg has to be both the right color and in the right position to earn a black scoring peg. When one of the code pegs (and one of the guess pegs) has resulted in a black scoring peg, eliminate them from the second phase,

scoring with the white pegs. White scoring pegs are awarded for pegs of the right color but in the wrong position. In practice, the guesser places his pegs in the row according to his guess relative to color and position. The scorer then places the small black and white pegs next to the row being scored. With these scoring pegs in mind, the guesser makes adjustments in his opinions regarding the code, and makes a new guess in the next row. This continues until the code has been guessed. Keep in mind that there can never be more than four scoring pegs awarded.

Example: The code word is Blue, White, White, Red. The guess is White, White, Red, Black. The scoring is as follows:

Code	Bl	Wh	Wh	Rd
Guess	Wh	Wh	Rd	Bl

Earns one black scoring peg.

During the first phase we examine each of the code and guess pegs looking for an exact match. The second peg position, white, falls into this category and it is the only peg position that does. The black scoring peg is therefore 1. We then move onto the second phase during which the white scoring pegs are awarded. Before doing this, any code and guess pegs are mentally eliminated from further consideration so they play no part in the white peg scoring. The other player must not be aware of which pegs are being eliminated from the scoring because he could then quickly guess the code. Half of his problem is in figuring out which pegs are earning the black scoring pegs and which are earning the white scoring pegs.

Code	Bl	XX	Wh	Rd
Guess	Wh	XX	Rd	Bl

These pairs each earn one white scoring peg.

The total score is thus 1 Black and 2 White scoring pegs.

THE MASTER-MIND PROGRAM. When playing the game of Master-Mind, you rather quickly discover that the fun is in breaking the code. The second player fills the role of creating the code word and then scoring the first player. The role of the second player can be filled quite well by the computer, which leaves to the user the fascinating job of trying to discover the codes created by the computer. The program we are about to describe is listed below and illustrated in the flow diagram of Fig. 11-2. Enter it into your computer following the flow diagram as you do so.

MASTER-MIND

```
0100    21    LXI    H, 001DH              ;Point H/L to first code register.
0101    1D                                 ;
0102    00                                 ;
0103    0E    MVI    C, 04H                ;Set loop counter for 4.
0104    04                                 ;
0105    CD    CALL RANDOM (0370H)          ;Generate random number from 0 to 5. Store
0106    70                                 ;the number in the code
0107    03                                 ;register. Increment the H/L
0108    05    RANDOM NUMBER RANGE          ;register pair to point to the next
0109    77    MOV    M, A                  ;code register. Decrement the loop
010A    23    INX    H                     ;counter and repeat if not the fourth
010B    0D    DCR    C                     ;random number.
010C    C2    JNZ    0105H                 ;
010D    05                                 ;
010E    01                                 ;
010F    21    LXI    H, 0021H              ;Point H/L to the first of the code
0110    21                                 ;image registers.
0111    00                                 ;
0112    11    LXI    D, 001DH              ;Point D/E to first of the code
0113    1D                                 ;registers. Transfer the contents
0114    00                                 ;of the code registers to the code
```

```
0115    0E    MVI    C, 04H               ;image registers.
0116    04                                ;
0117    1A    LDAX   D                    ;
0118    77    MOV    M, A                 ;
0119    13    INX    D                    ;
011A    23    INX    H                    ;
011B    0D    DCR    C                    ;
011C    C2    JNZ    0117H                ;
011D    17                                ;
011E    01                                ;
011F    CD    CALL BLANK (80CCH)          ;Blank displays.
0120    CC                                ;
0121    80                                ;
0122    CD    CALL CONVERT (8132H)        ;Convert contents of display register
0123    32                                ;into 7-segment codes in display
0124    81                                ;buffers.
0125    CD    CALL KEYDSPLY (80F2H)       ;Display contents of buffers and
0126    F2                                ;wait for a key closure.  If this
0127    80                                ;closure is NEXT, go to 0133H.
0128    FE    CPI NEXT (12H)              ;
0129    12                                ;
012A    CA    JZ     0133H                ;
012B    33                                ;
012C    01                                ;
012D    CD    CALL SUB1B (8072H)          ;This point has been reached because
012E    72                                ;a key closure other than NEXT has
012F    80                                ;been found.  Shift the key closure
0130    C3    JMP    0122H                ;into the displays and go back to
0131    22                                ;0122H to display and wait for another
0132    01                                ;key closure.
0133    11    LXI    D, 0025H             ;This point has been reached because
0134    25                                ;the NEXT key was pressed.  It is
0135    00                                ;time to process the guess in the
```

```
0136    21    LXI    H, DIRG3 (000BH)   ;display registers.  Load the
0137    0B                              ;guess into the guess registers.
0138    00                              ;
0139    0E    MVI    C, 04H             ;
013A    04                              ;
013B    7E    MOV    A, M               ;
013C    12    STAX   D                  ;
013D    23    INX    H                  ;
013E    13    INX    D                  ;
013F    0D    DCR    C                  ;
0140    C2    JNZ    013BH              ;
0141    3B                              ;
0142    01                              ;
0143    00    NOP                       ;After the guess registers are loaded
0144    3E    MVI    A, 00H             ;it is time to score the guess.  First
0145    00                              ;we zero out the black scores at 002AH
0146    32    STA    002AH              ;and the white scores at 002BH.
0147    2A                              ;
0148    00                              ;
0149    32    STA    002BH              ;
014A    2B                              ;
014B    00                              ;
014C    11    LXI    D, 0025H           ;Score the blacks.  Point D/E to
014D    25                              ;first of guess registers and H/L to
014E    00                              ;first of code image registers.
014F    21    LXI    H, 0021H           ;Move code image register to B.  Is
0150    21                              ;this the last of the image registers?
0151    00                              ;If so, go to 016EH to continue.  If
0152    46    MOV    B, M               ;not, load the accumulator with the
0153    7D    MOV    A, L               ;contents of the guess register.
0154    FE    CPI    25H                ;Compare to code in B.  If it is not
0155    25                              ;a match, go to 0169H to continue.  If
```

```
0156   CA   JZ     016EH      ;a match is found load FFH into both
0157   6E                     ;the guess register and the code image
0158   01                     ;register.  Then increment the black
0159   1A   LDAX   D          ;scoring register at 002AH.
015A   B8   CMP    B          ;
015B   C2   JNZ    0169H      ;
015C   69                     ;
015D   01                     ;
015E   3E   MVI    A, FFH     ;
015F   FF                     ;
0160   12   STAX   D          ;
0161   77   MOV    M, A       ;
0162   3A   LDA    002AH      ;
0163   2A                     ;
0164   00                     ;
0165   3C   INR    A          ;
0166   32   STA    002AH      ;
0167   2A                     ;
0168   00                     ;
0169   13   INX    D          ;Point D/E to next guess register and
016A   23   INX    H          ;H/L to next code image register.
016B   C3   JMP    0152H      ;Go back and check the next code
016C   52                     ;and guess registers.
016D   01                     ;
016E   00   NOP               ;This point has been reached because
016F   11   LXI    D, 0025H   ;it is time to score the white pegs.
0170   25                     ;Point the D/E registers to the first
0171   00                     ;of the guess registers and H/L to
0172   21   LXI    H, 0021H   ;the first of the code image registers.
0173   21                     ;Then load the code image register
0174   00                     ;into the B register.  Does it
0175   46   MOV    B, M       ;contain FFH?  If so, go to 0198H
```

```
0176    7E    MOV    A, M       ;If not, load the accumulator with
0177    FE    CPI    FFH        ;the contents of the guess register
0178    FF                      ;and compare to the code register.
0179    CA    JZ     0198H      ;If there is no match, go to 018FH.
017A    98                      ;
017B    01                      ;
017C    1A    LDAX   D          ;
017D    B8    CMP    B          ;
017E    C2    JNZ    018FH      ;
017F    8F                      ;
0180    01                      ;
0181    3E    MVI    A, FFH     ;If there is a match, load both
0182    FF                      ;registers with FFH, increment the
0183    12    STAX   D          ;white counter at 002BH and
0184    77    MOV    M, A       ;go on to 0196H.
0185    3A    LDA    002BH      ;
0186    2B                      ;
0187    00                      ;
0188    3C    INR    A          ;
0189    32    STA    002BH      ;
018A    2B                      ;
018B    00                      ;
018C    C3    JMP    0196H      ;
018D    96                      ;
018E    01                      ;
018F    13    INX    D          ;This point is reached because there
0190    7B    MOV    A, E       ;was no match.  Point D/E at the next
0191    FE    CPI    29H        ;guess register.  Is it the last?
0192    29                      ;If not, go to 0175H and repeat
0193    C2    JNZ    0175H      ;the loop.  If it was the last,
0194    75                      ;start the guess pointer back at the
0195    01                      ;first register,  increment the code
```

```
0196   1E   MVI   E, 25H           ;pointer and take another pass through
0197   25                          ;the registers.  Each time a match
0198   23   INX   H                ;is made, both registers are loaded
0199   7D   MOV   A, L             ;with FFH,
019A   FE   CPI   25H              ;increment the white counter, set the
019B   25                          ;guess pointer back at the first
019C   C2   JNZ   0175H            ;guess register at 0025H and start
019D   75                          ;over.
019E   01                          ;
019F   3A   LDA   002AH            ;Store the contents of the black
01A0   2A                          ;register at 0009H, Display
01A1   00                          ;Register no. 1.
01A2   32   STA   0009H            ;
01A3   09                          ;
01A4   00                          ;
01A5   3A   LDA   002BH            ;Store the contents of the white
01A6   2B                          ;register at 0008H, Display Register
01A7   00                          ;no. 0.
01A8   32   STA   0008H            ;
01A9   08                          ;
01AA   00                          ;
01AB   CD   CALL CONVERT (8132H)   ;Convert the contents of the display
01AC   32                          ;registers into their 7-segment
01AD   81                          ;equivalents in the display buffers.
01AE   CD   CALL KEYDSPLY (80F2H)  ;Display the score along with the
01AF   F2                          ;guess.  Ignore all key closures
01B0   80                          ;except NEXT.  When NEXT is found,
01B1   FE   CPI NEXT (12H)         ;go back to 010FH to shift in next
01B2   12                          ;guess.
01B3   C2   JNZ   01B2H            ;
01B4   B2                          ;
01B5   01                          ;
01B6   C3   JMP   010FH            ;
01B7   0F                          ;
01B8   01                          ;
```

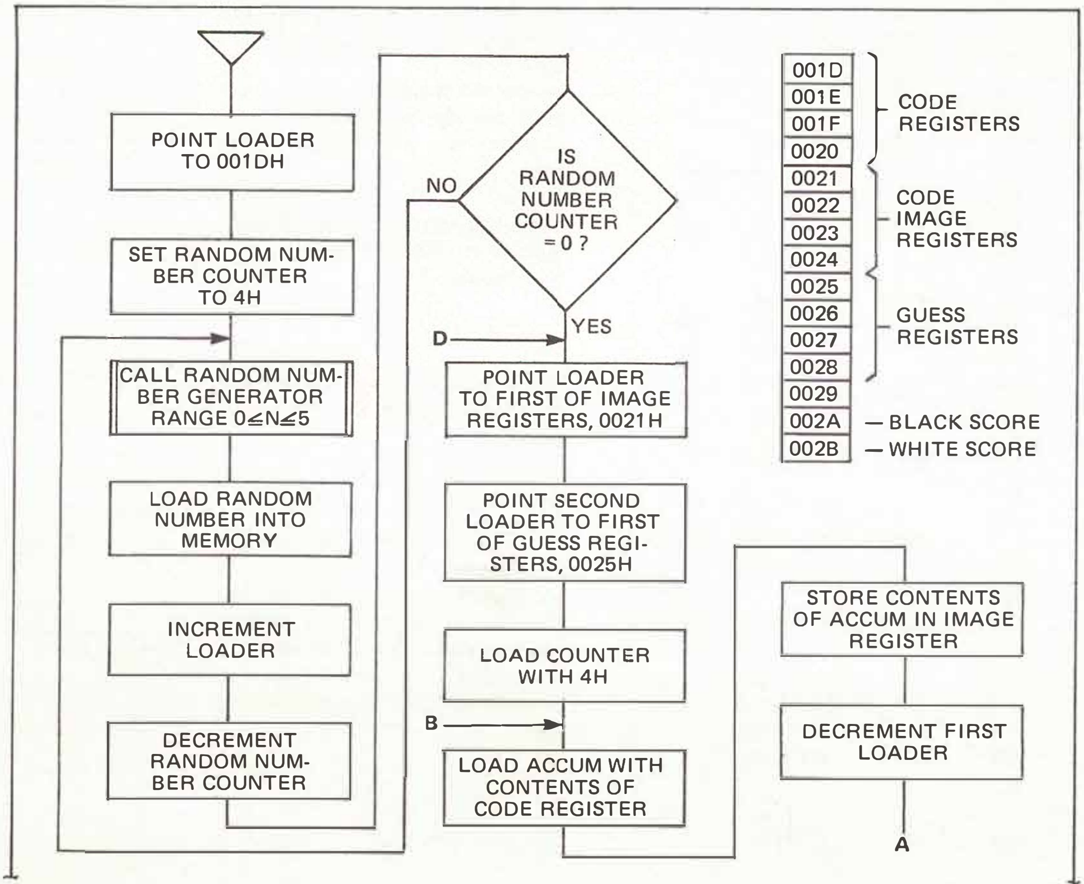
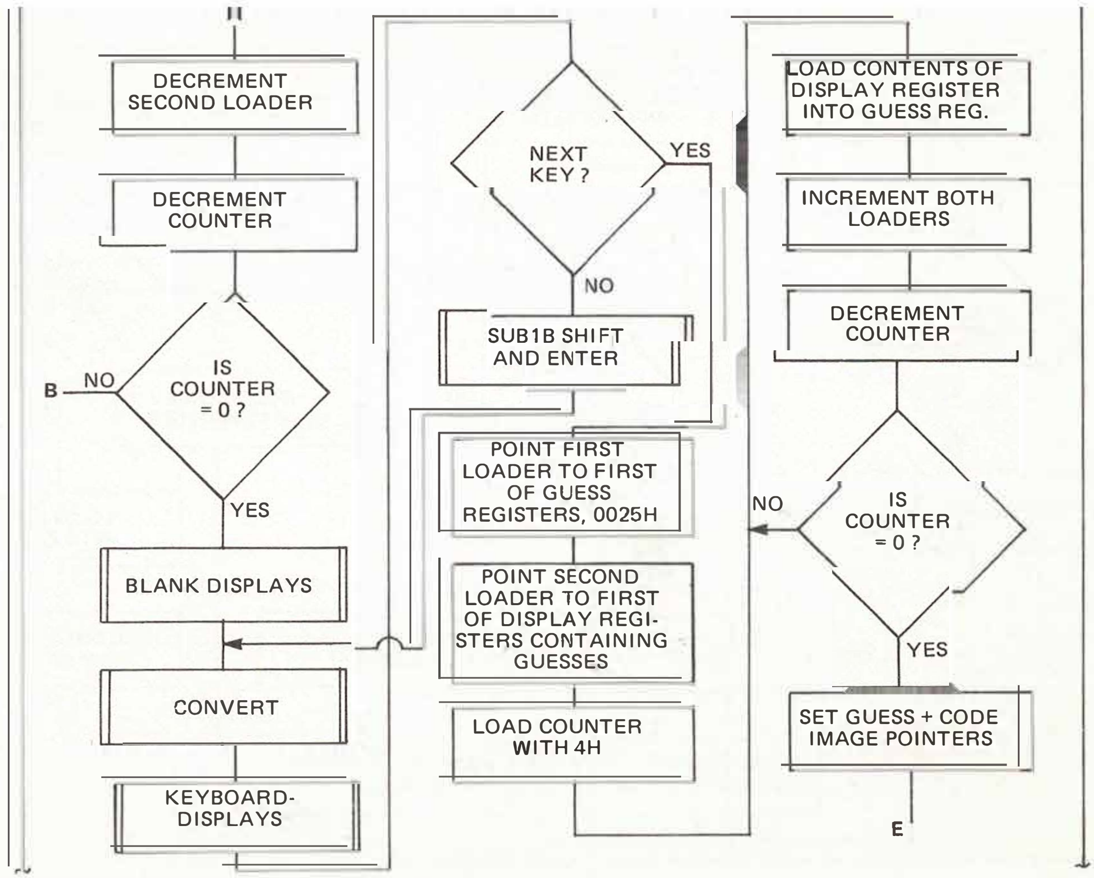
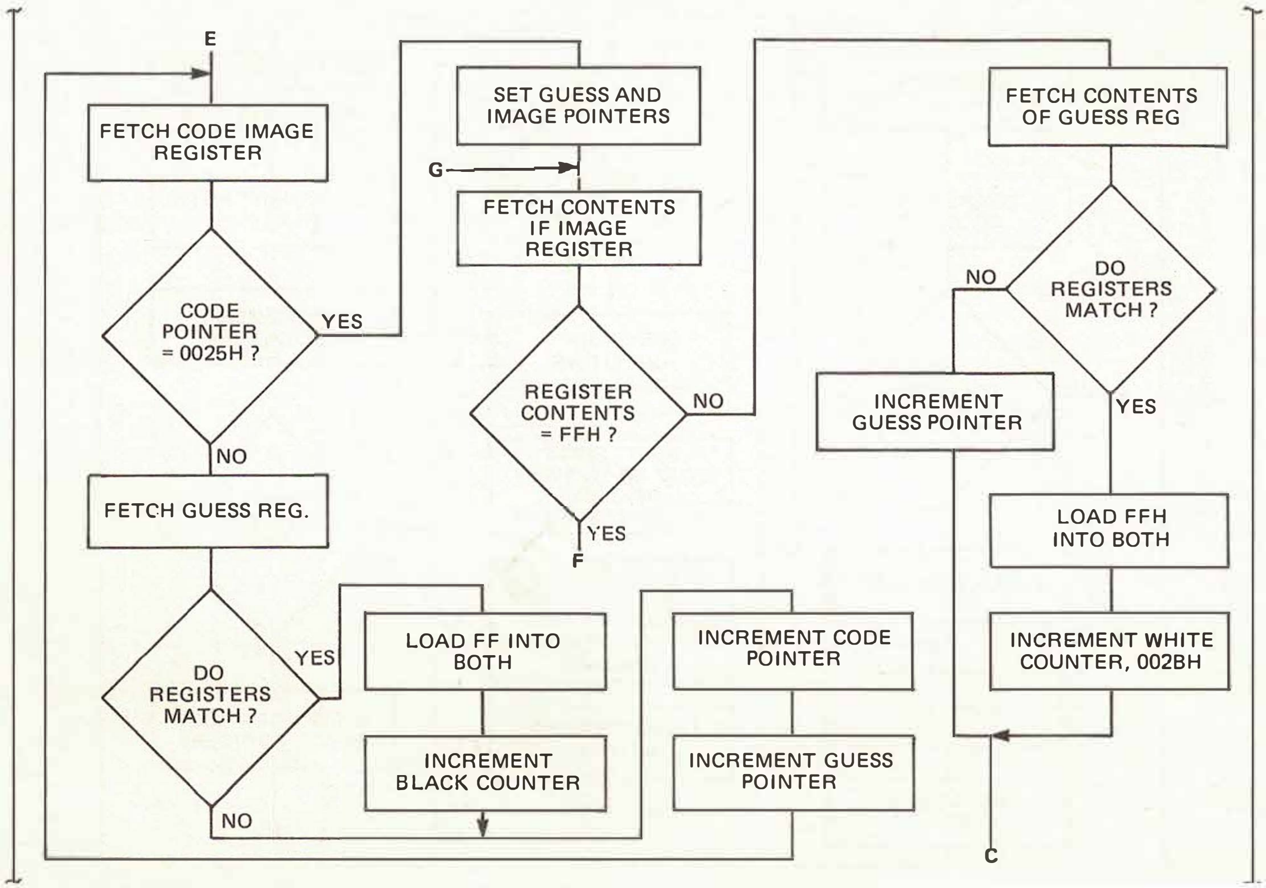
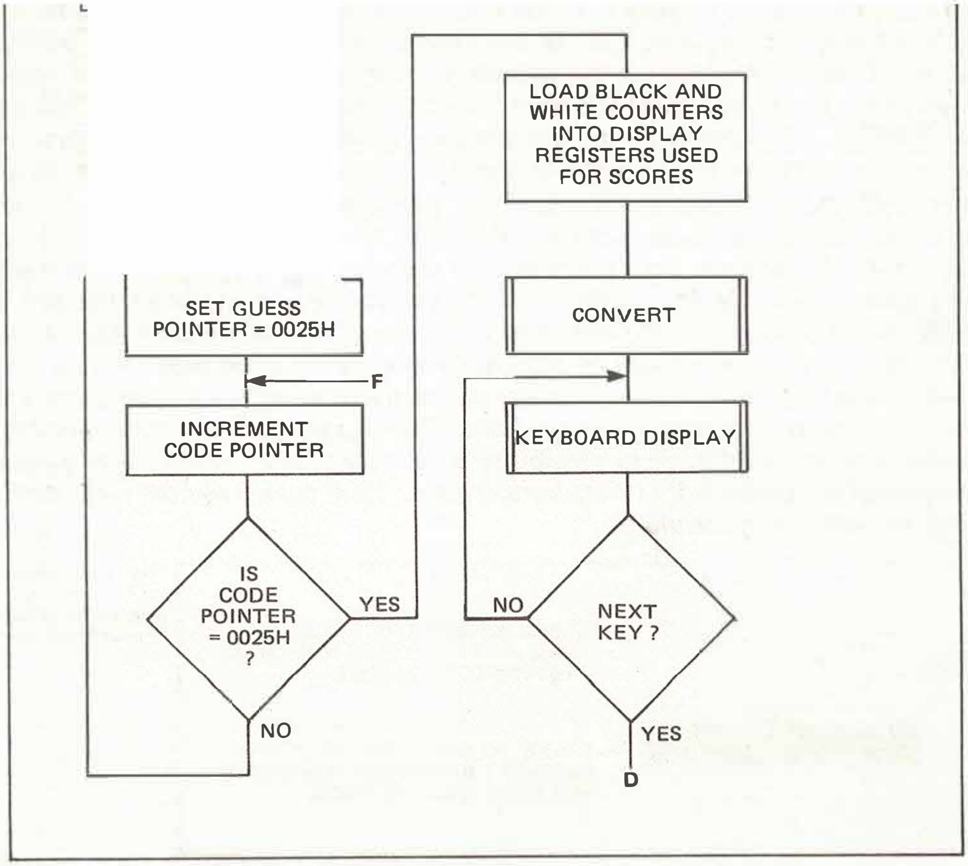


Fig. 11-2 Flow diagram of the Master-Mind program.



DECREMENT SECOND LOADER
DECREMENT COUNTER
B
NO
IS COUNTER = 0 ?
YES
BLANK DISPLAYS
CONVERT
KEYBOARD-DISPLAYS
NEXT KEY ?
YES
NO
SUB1B SHIFT AND ENTER
POINT FIRST LOADER TO FIRST OF GUESS REGISTERS, 0025H
POINT SECOND LOADER TO FIRST OF DISPLAY REGISTERS CONTAINING GUESSES
LOAD COUNTER WITH 4H
LOAD CONTENTS OF DISPLAY REGISTER INTO GUESS REG.
INCREMENT BOTH LOADERS
DECREMENT COUNTER
NO
IS COUNTER = 0 ?
YES
SET GUESS + CODE IMAGE POINTERS
E


E
FETCH CODE IMAGE REGISTER
CODE POINTER = 0025H ?
YES
NO
FETCH GUESS REG.
DO REGISTERS MATCH ?
YES
NO
LOAD FF INTO BOTH
INCREMENT BLACK COUNTER
SET GUESS AND IMAGE POINTERS
G
FETCH CONTENTS IF IMAGE REGISTER
REGISTER CONTENTS = FFH ?
NO
YES
F
INCREMENT CODE POINTER
INCREMENT GUESS POINTER
FETCH CONTENTS OF GUESS REG
DO REGISTERS MATCH ?
NO
YES
INCREMENT GUESS POINTER
LOAD FFH INTO BOTH
INCREMENT WHITE COUNTER, 002BH
C


LOAD BLACK AND WHITE COUNTERS INTO DISPLAY REGISTERS USED FOR SCORES
SET GUESS POINTER = 0025H
F
INCREMENT CODE POINTER
CONVERT
KEYBOARD DISPLAY
IS CODE POINTER = 0025H ?
YES
NO
NEXT KEY ?
NO
YES
D

To use the program, press CLR and EXC. The computer immediately makes up a code. Make your first guess by pressing the numerals 0-5. If you change your mind, just keep pressing the correct combination of keys. As you do, the incorrect entry will be shifted out and replaced with the correct version. When four numerals appear in the displays and you are satisfied with your guess, press NEXT. The computer will process your guess and display your score on the right. The first scoring digit represents the black scoring pegs, that is, the number of guess numerals that were both the right numeral and the right position. The second digit is the white scoring peg, that is, the number of guess numerals that were the correct numeral but the wrong position. See Fig. 11-3. Write down your guess and the score it earned. When you are ready to enter your second guess, press NEXT to clear out the previous guess and enter the new one. Again press NEXT to see your score. By repeating this process, carefully examining the results of previous guesses as you go, you should be able to determine the code word in four or five guesses. One word of strategy . . . It is usually better not to try to eliminate the variables one by one by using known numerals in three of the positions. This approach takes many guesses. A much better strategy is to realize that every possibility is as likely as every other. Any guess that fits the data from the earlier guesses is therefore a good guess. Each guess provides more data and reduces the number of legitimate code words.

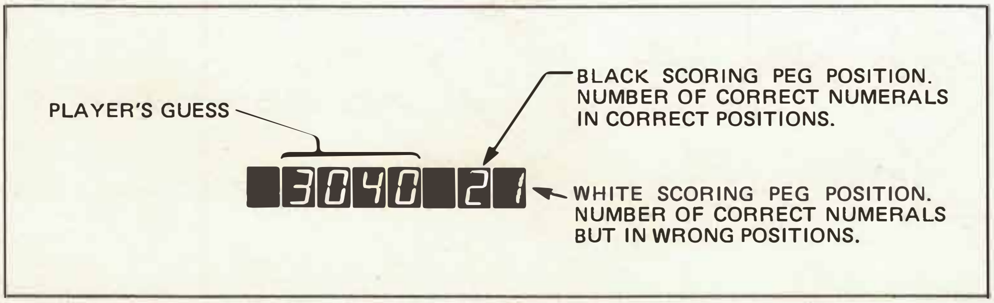


Fig. 11-3 Guess and scoring display for ia7301 Master Mind game.